



arm

Debugging and Profiling HPC Applications with Arm Forge

ATPESC

August 5, 2020

Agenda

- General Debugging and Profiling Advice
- Arm Software for Debugging and Profiling
- Debugging with DDT
- Profiling with MAP
- Theta Specific Settings

Debugging

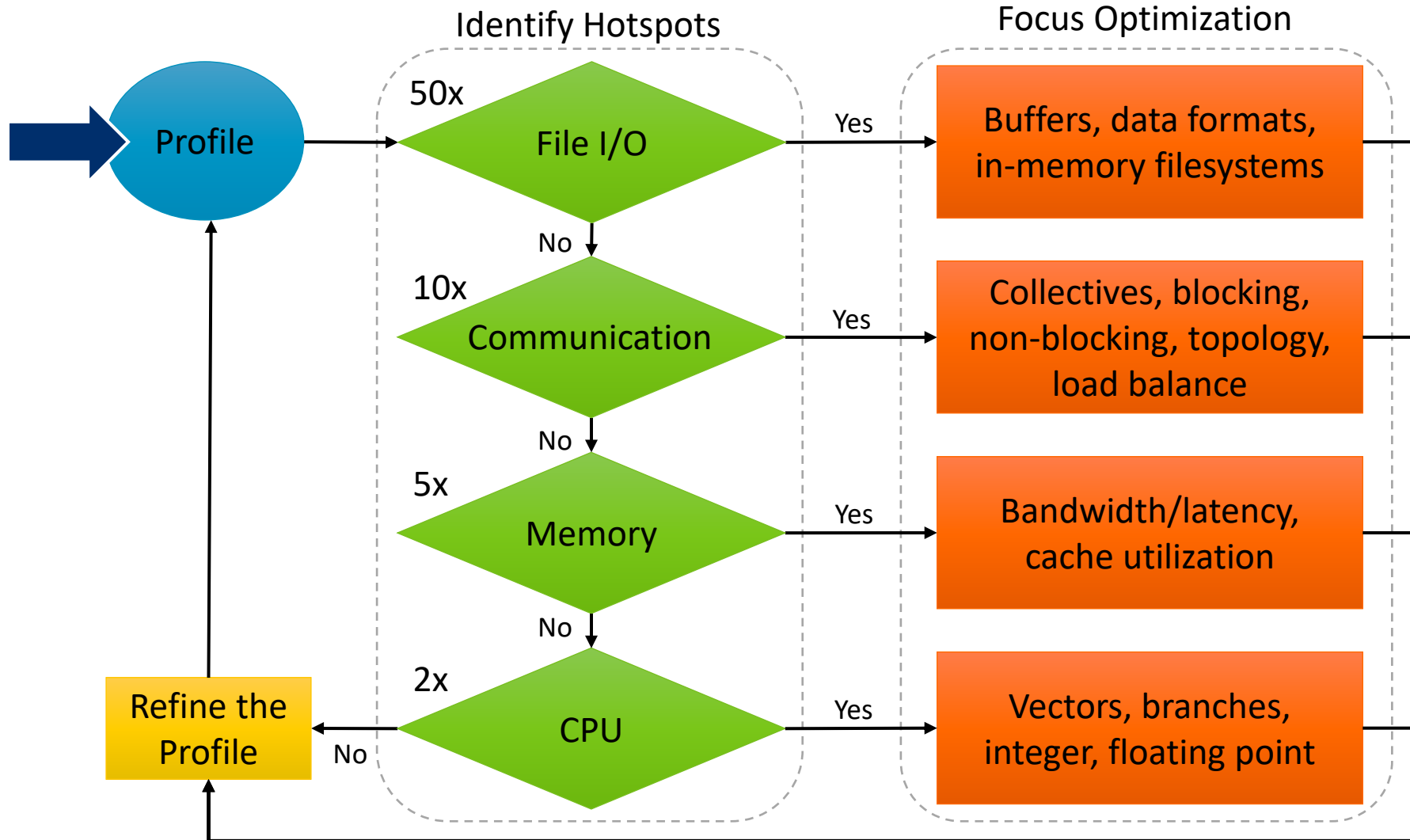
Transforming a broken program to a working one

How? TRAFFIC!

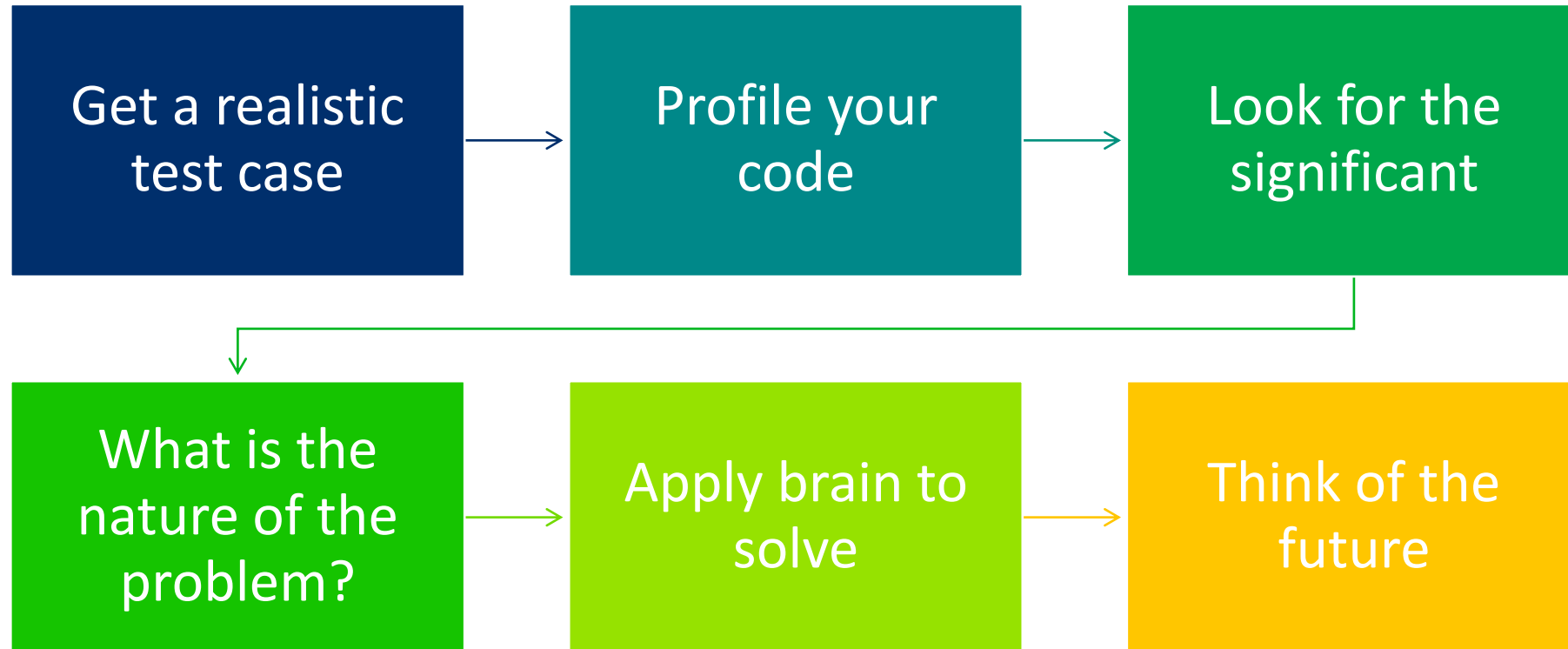
- **T**rack the problem
- **R**eproduce
- **A**utomate - (and simplify) the test case
- **F**ind origins – where could the “infection” be from?
- **F**ocus – examine the origins
- **I**solate – narrow down the origins
- **C**orrect – fix and verify the test case is successful

Profiling

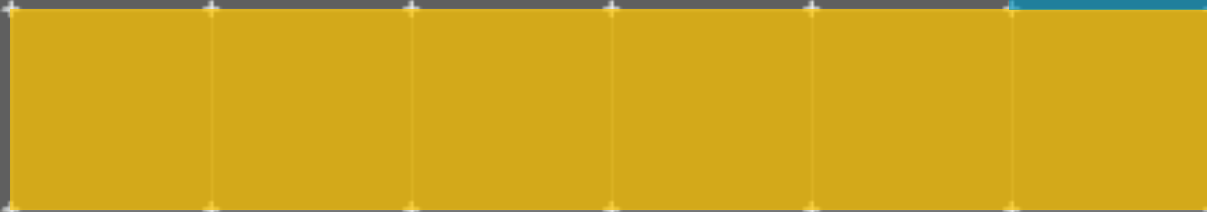
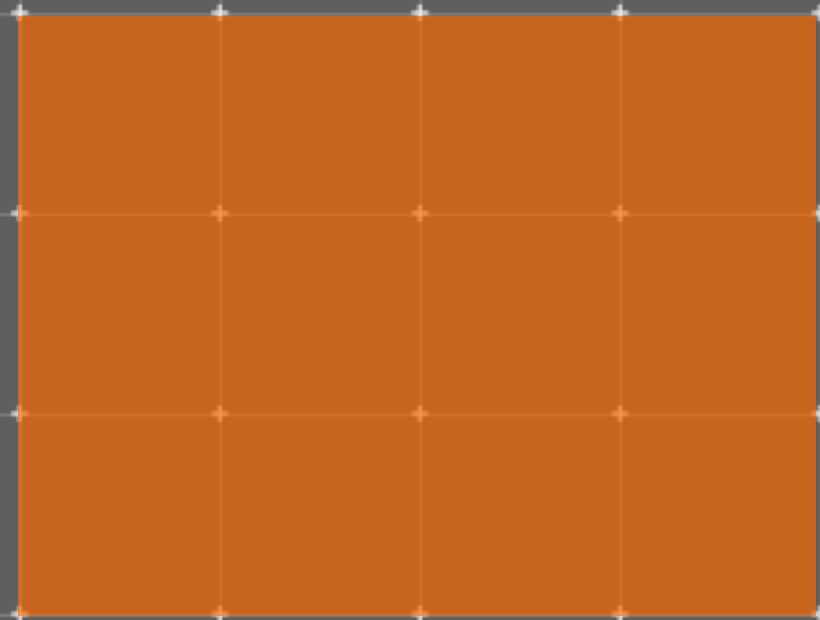
Profiling is central to understanding and improving application performance.



Performance Improvement Workflow



Arm Software



Arm Forge

An interoperable toolkit for debugging and profiling



Commercially supported
by Arm



Fully Scalable



Very user-friendly

The de-facto standard for HPC development

- Available on the vast majority of the Top500 machines in the world
- Fully supported by Arm on x86, IBM Power, Nvidia GPUs, etc.

State-of-the art debugging and profiling capabilities

- Powerful and in-depth error detection mechanisms (including memory debugging)
- Sampling-based profiler to identify and understand bottlenecks
- Available at any scale (from serial to parallel applications running at petascale)

Easy to use by everyone

- Unique capabilities to simplify remote interactive sessions
- Innovative approach to present quintessential information to users

Run and ensure application correctness

Combination of debugging and re-compilation

- Ensure application correctness with **Arm DDT scalable debugger**
- Integrate with continuous integration system.
- Use version control to track changes and leverage Forge's built-in VCS support.

Examples:

```
$> ddt --offline aprun -n 48 ./example
```

```
$> ddt --connect aprun -n 48 ./example
```

15		2:17.256	0-7	Play				
16		2:18.048	4-7	Process stopped at breakpoint in main (cpi.c:50).				
17				Additional Information <div>▼ Stacks</div> <table><thead><tr><th>Processes</th><th>Function</th></tr></thead><tbody><tr><td>4-7</td><td>main (cpi.c:50)</td></tr></tbody></table>	Processes	Function	4-7	main (cpi.c:50)
Processes	Function							
4-7	main (cpi.c:50)							
18		2:19.048	n/a	Select process 4				
19				Additional Information <div>► Current Stack</div> <div>► Locals</div>				

Values

numprocs: — 8 myid: from 0 to 7 n: — 100

umprocs: — 8 myid: from 0 to 7 n: — 100

numprocs: — 8 myid: from 0 to 7 n: — 100

numprocs: — 8 myid: from 0 to 7 n: — 100

numprocs: — 8 myid: from 0 to 7 n: — 100

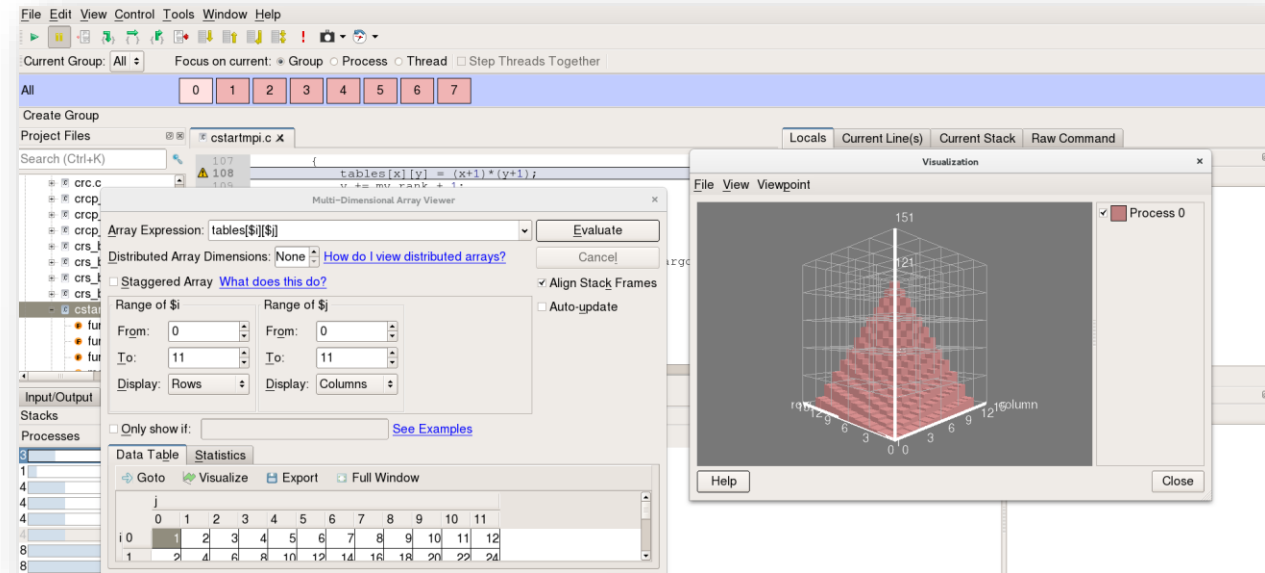
numprocs: — 8 myid: from 0 to 7 n: — 100

numprocs: — 8 myid: from 0 to 7 n: — 100

numprocs: — 8 myid: from 0 to 7 n: — 100

9	2:17.832	main (cpi.c:46)	0-7	done: — 0 i: from 65 to 72 numprocs: — 8 myid: from 0 to 7 n: — 100
10	2:17.832	main (cpi.c:46)	0-7	done: — 0 i: from 73 to 80 numprocs: — 8 myid: from 0 to 7 n: — 100
11	2:18.323	main (cpi.c:46)	0-7	done: — 0 i: from 81 to 88 numprocs: — 8 myid: from 0 to 7 n: — 100
12	2:18.323	main (cpi.c:46)	0-7	done: — 0 i: from 89 to 96 numprocs: — 8 myid: from 0 to 7 n: — 100
13	2:18.325	main (cpi.c:46)	0-3	done: — 0 i: from 97 to 100 numprocs: — 8 myid: from 0 to 3 n: — 100

8 © 2018 Arm

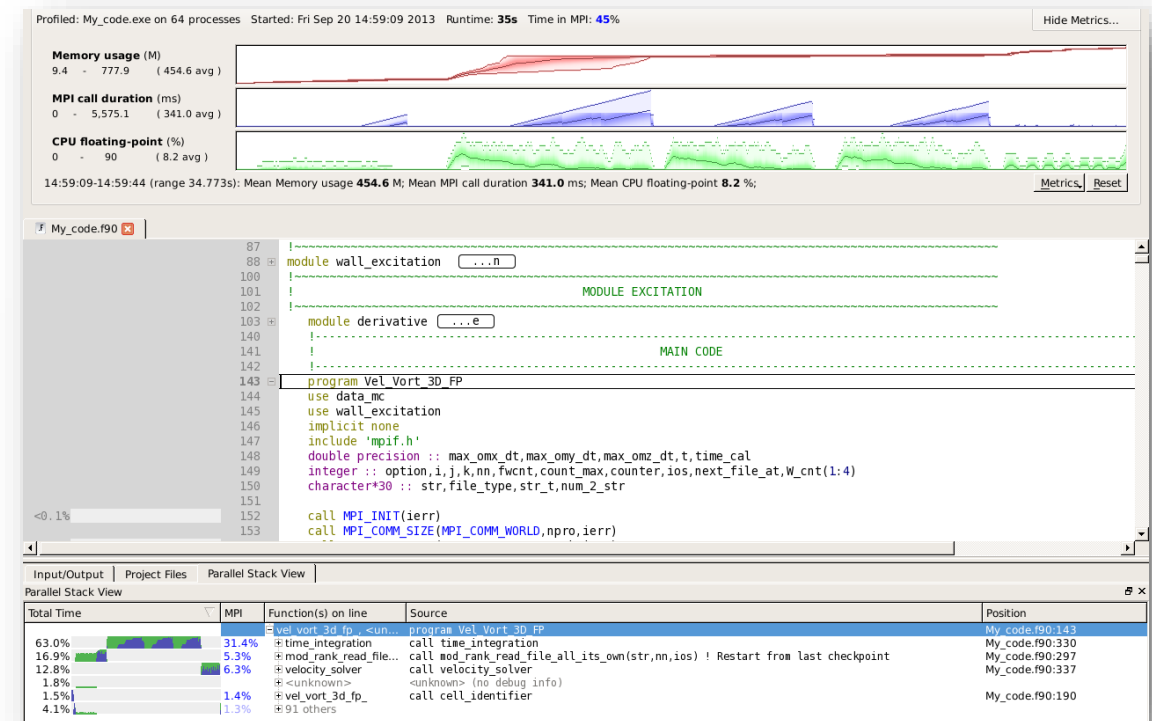
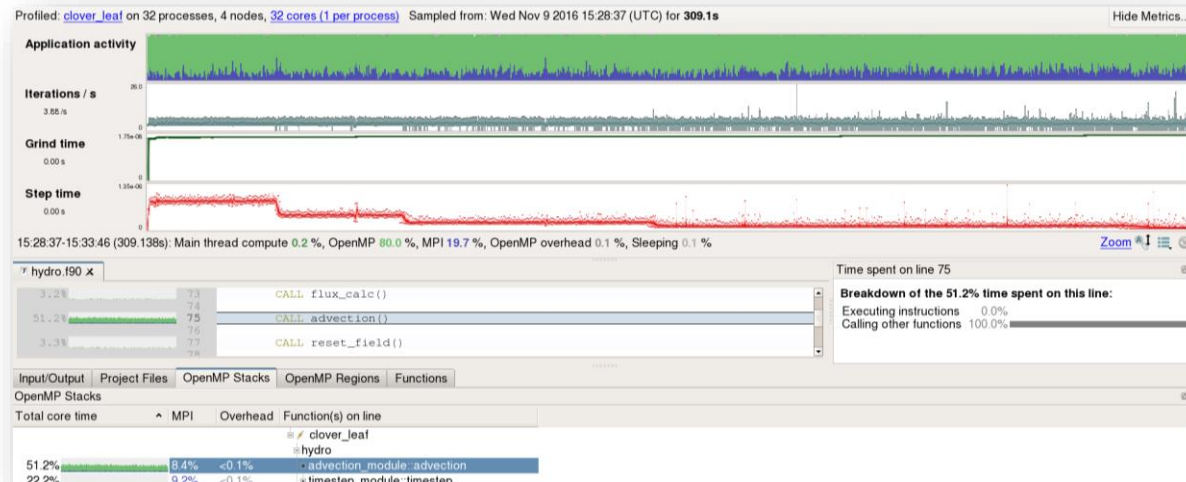


Visualize the performance of your application

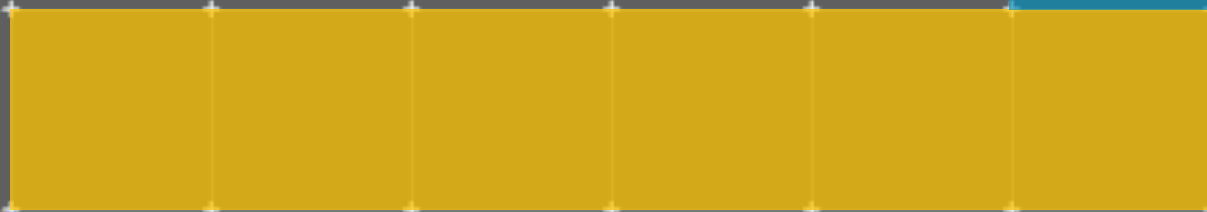
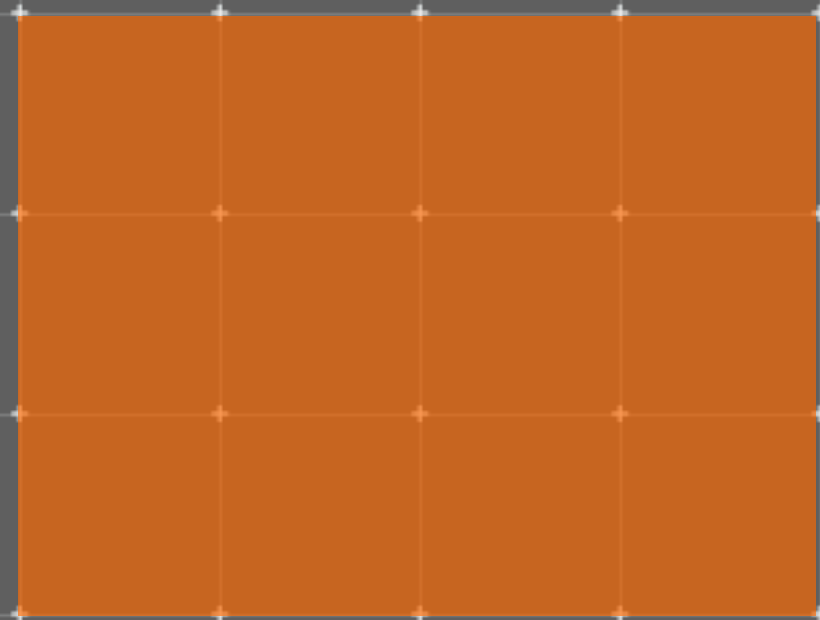
- Measure all performance aspects with **Arm MAP parallel profiler**
- Identify bottlenecks and rewrite some code for better performance

Examples:

```
$> map --profile -n 48 ./example
```



Debugging with DDT



Arm DDT – The Debugger

Who had a rogue behaviour ?

- Merges stacks from processes and threads

Where did it happen?

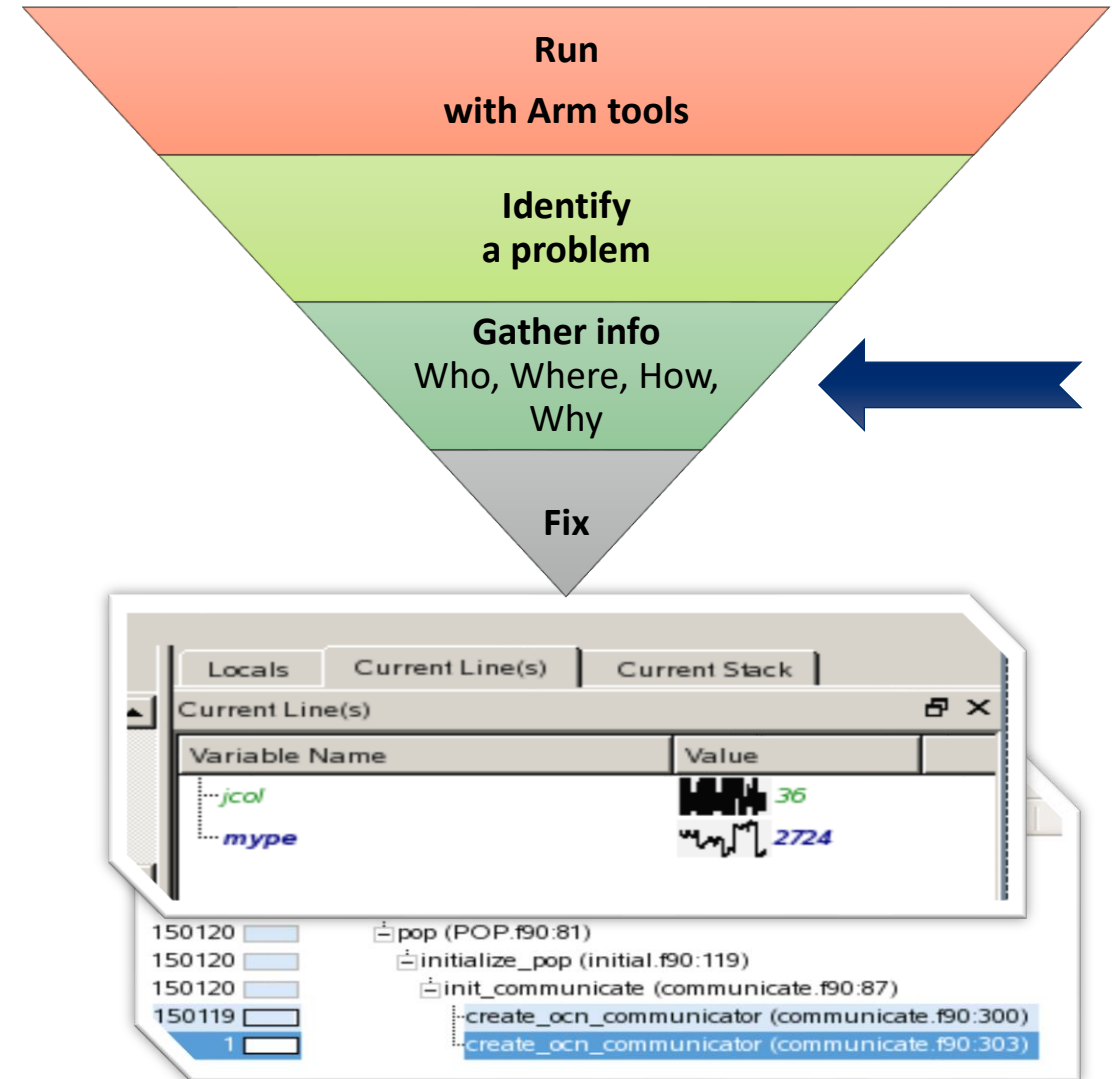
- leaps to source

How did it happen?

- Diagnostic messages
- Some faults evident instantly from source

Why did it happen?

- Unique “Smart Highlighting”
- Sparklines comparing data across processes



Preparing Code for Use with DDT

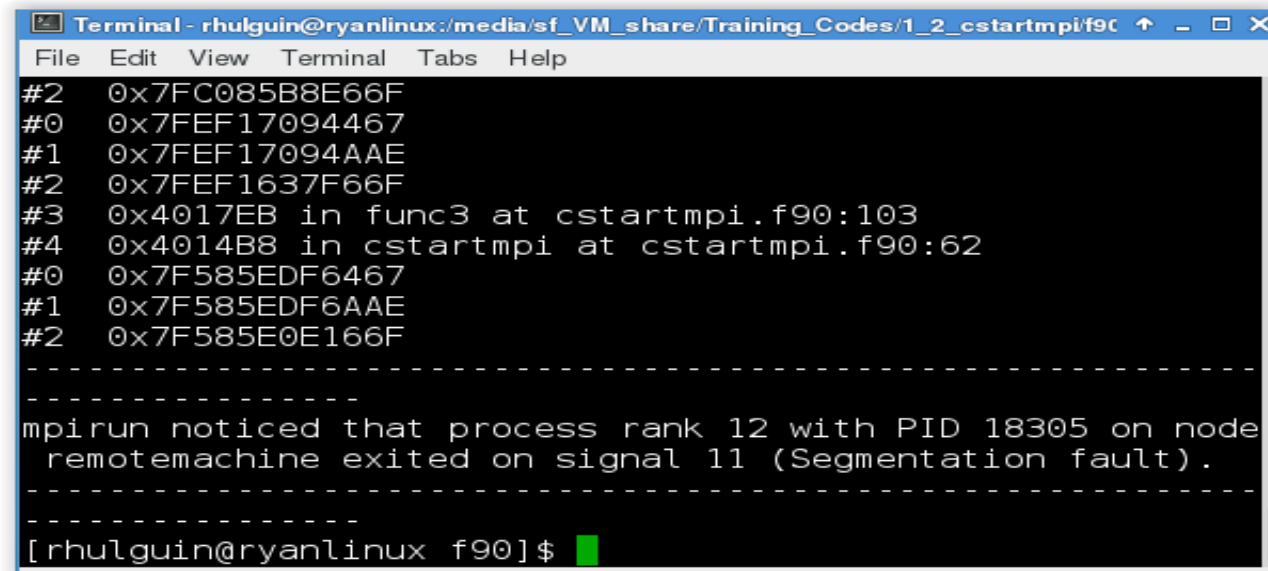
As with any debugger, code must be compiled with the debug flag typically `-g`

It is recommended to turn off optimization flags i.e. `-O0`

Leaving optimizations turned on can cause the compiler to *optimize out* some variables and even functions making it more difficult to debug

Segmentation Fault

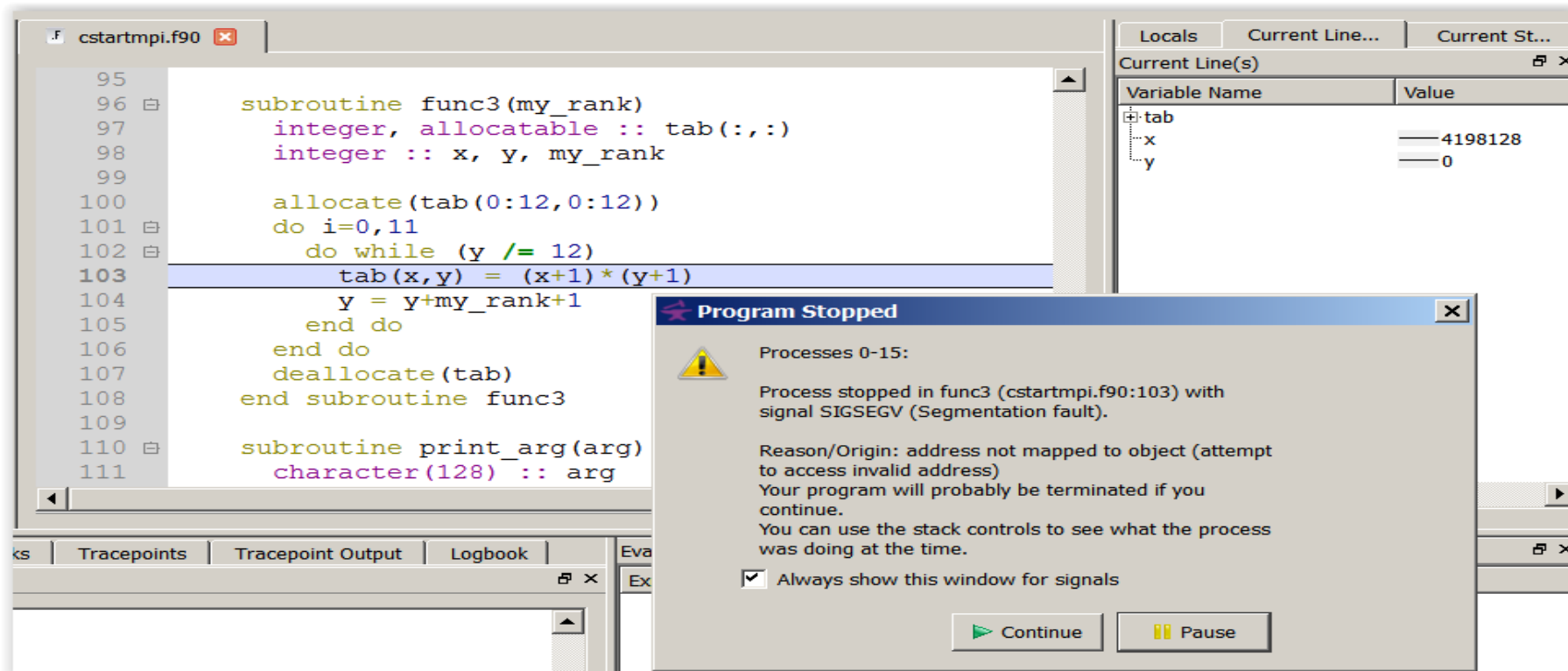
In this example, the application crashes with a segmentation error outside of DDT.

A terminal window titled "Terminal - rhulguin@ryanlinux:/media/sf_VM_share/Training_Codes/1_2_cstartmpi/f90" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal output shows a list of memory addresses and a message from mpirun: "mpirun noticed that process rank 12 with PID 18305 on node remotemachine exited on signal 11 (Segmentation fault)." followed by a dashed line and the prompt "[rhulguin@ryanlinux f90]\$".

```
Terminal - rhulguin@ryanlinux:/media/sf_VM_share/Training_Codes/1_2_cstartmpi/f90
File Edit View Terminal Tabs Help
#2 0x7FC085B8E66F
#0 0x7FEF17094467
#1 0x7FEF17094AAE
#2 0x7FEF1637F66F
#3 0x4017EB in func3 at cstartmpi.f90:103
#4 0x4014B8 in cstartmpi at cstartmpi.f90:62
#0 0x7F585EDF6467
#1 0x7F585EDF6AAE
#2 0x7F585E0E166F
-----
mpirun noticed that process rank 12 with PID 18305 on node
remotemachine exited on signal 11 (Segmentation fault).
-----
[rhulguin@ryanlinux f90]$
```

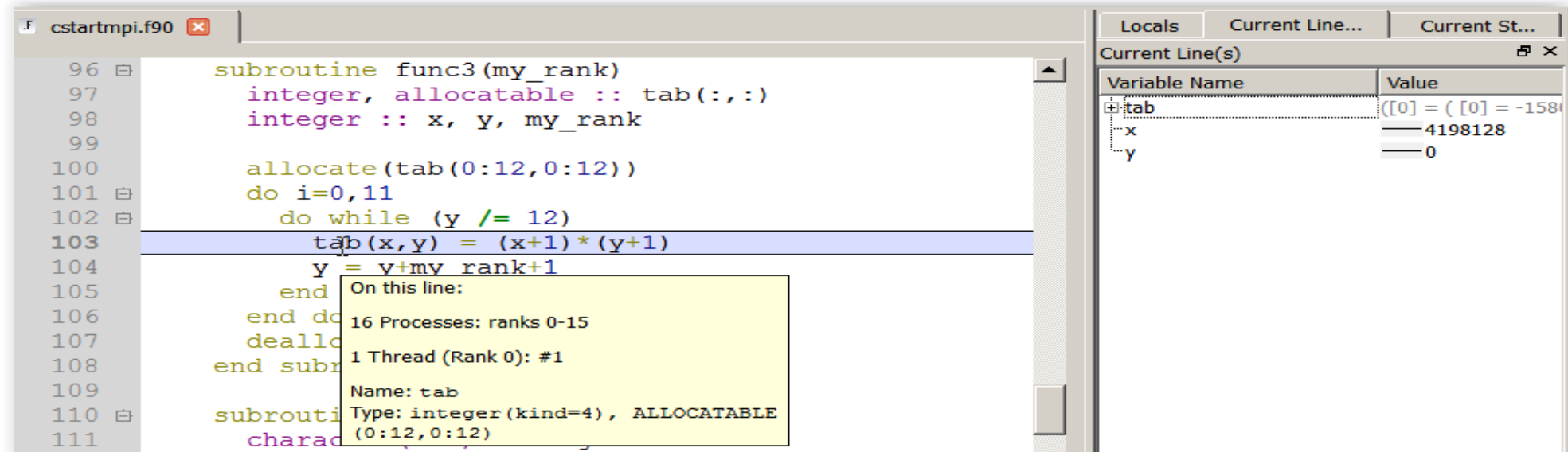
What happens when it runs under DDT?

Segmentation Fault in DDT



DDT takes you to the exact line where Segmentation fault occurred, and you can pause and investigate

Invalid Memory Access



```
103  tab(x,y) = (x+1)*(y+1)
```

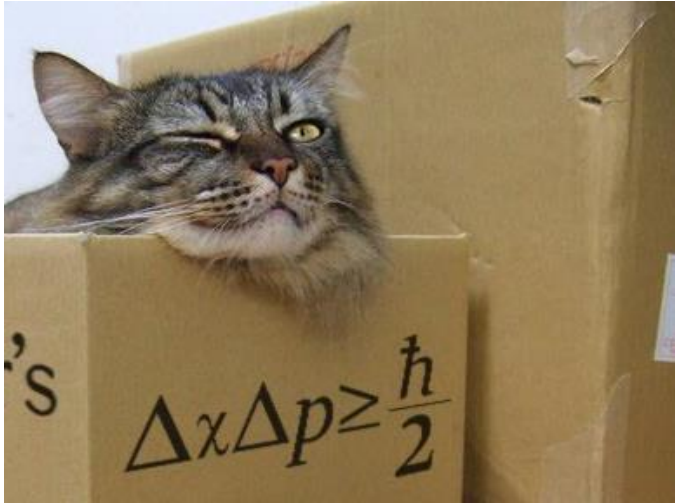
On this line:
16 Processes: ranks 0-15
1 Thread (Rank 0): #1
Name: tab
Type: integer(kind=4), ALLOCATABLE
(0:12,0:12)

Variable Name	Value
tab	[[0] = ([0] = -158
x	4198128
y	0

The array `tab` is a 13x13 array, but the application is trying to write a value to `tab(4198128,0)` which causes the segmentation fault.

`i` is not used, and `x` and `y` are not initialized

It works... Well, most of the time



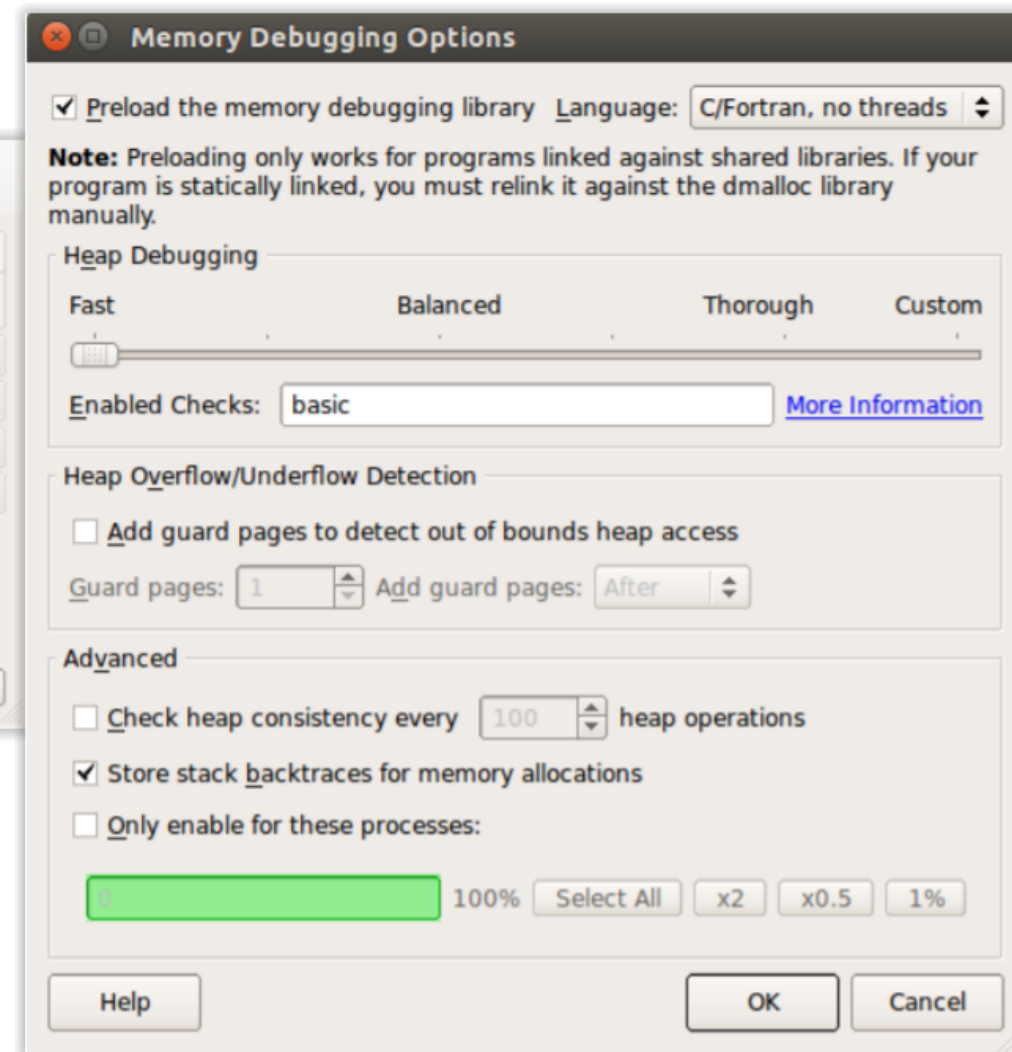
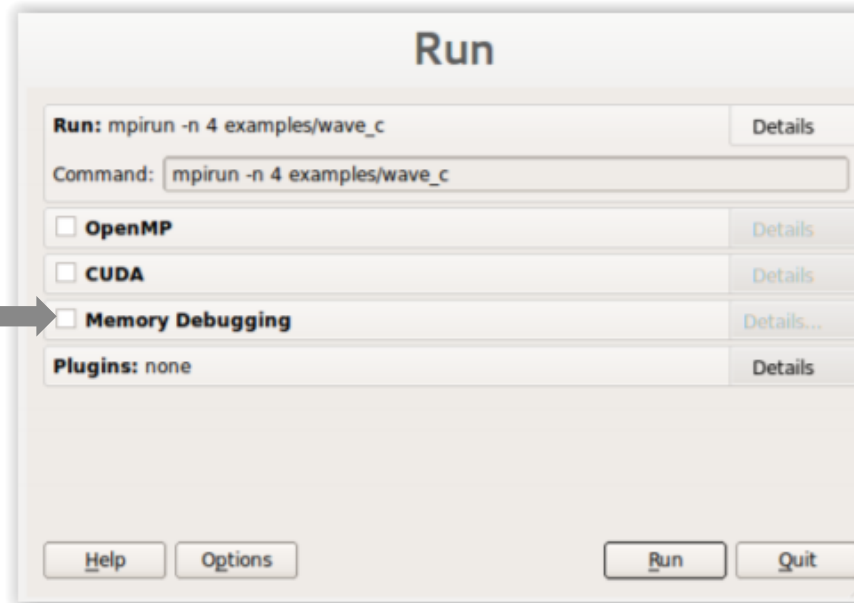
**SCHRODIN
BUG**



A strange behaviour where the application “sometimes” crashes is a typical sign of a memory bug

Arm DDT is able to force the crash to happen

Advanced Memory Debugging



Heap debugging options available

Fast

basic

- Detect invalid pointers passed to memory functions (e.g. malloc, free, ALLOCATE, DEALLOCATE,...)

check-fence

- Check the end of an allocation has not been overwritten when it is freed.

free-protect

- Protect freed memory (using hardware memory protection) so subsequent read/writes cause a fatal error.

Added goodness

- Memory usage, statistics, etc.

Balanced

free-blank

- Overwrite the bytes of freed memory with a known value.

alloc-blank

- Initialise the bytes of new allocations with a known value.

check-heap

- Check for heap corruption (e.g. due to writes to invalid memory addresses).

realloc-copy

- Always copy data to a new pointer when re-allocating a memory allocation (e.g. due to realloc)

Thorough

check-blank

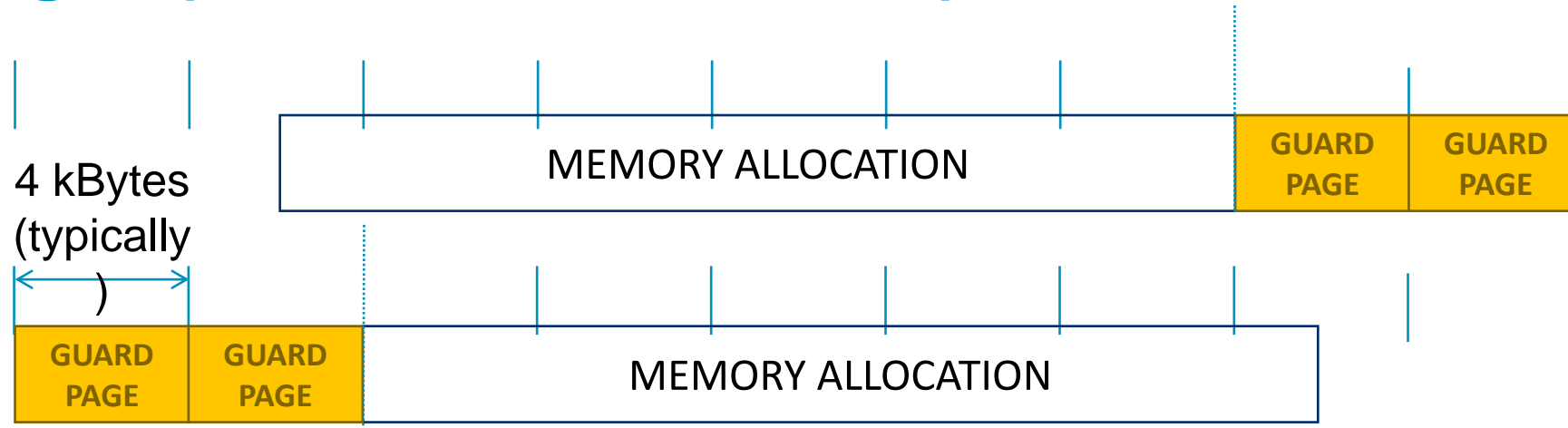
- Check to see if space that was blanked when a pointer was allocated/freed has been overwritten.

check-funcs

- Check the arguments of addition functions (mostly string operations) for invalid pointers.

*See user-guide:
Chapter 12.3.2*

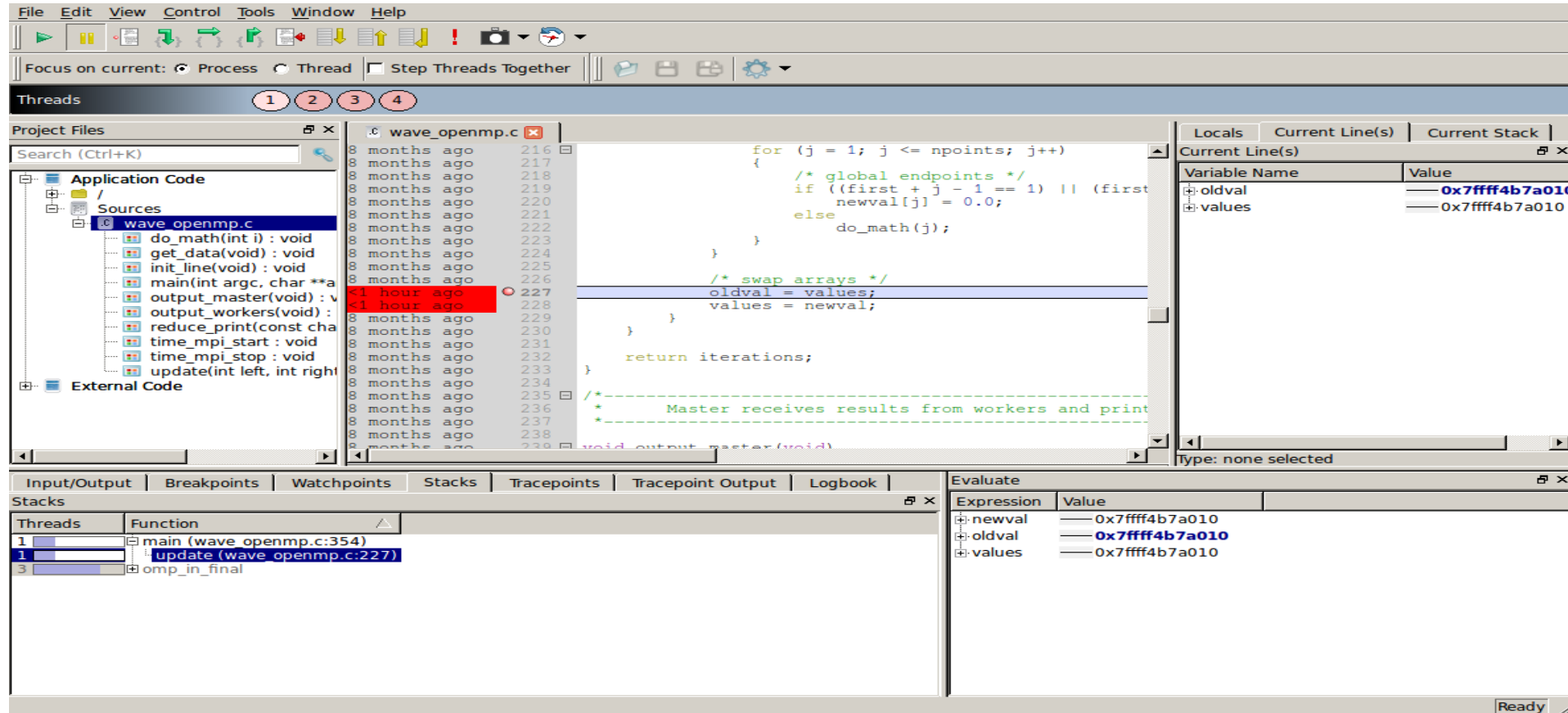
Guard pages (aka “Electric Fences”)



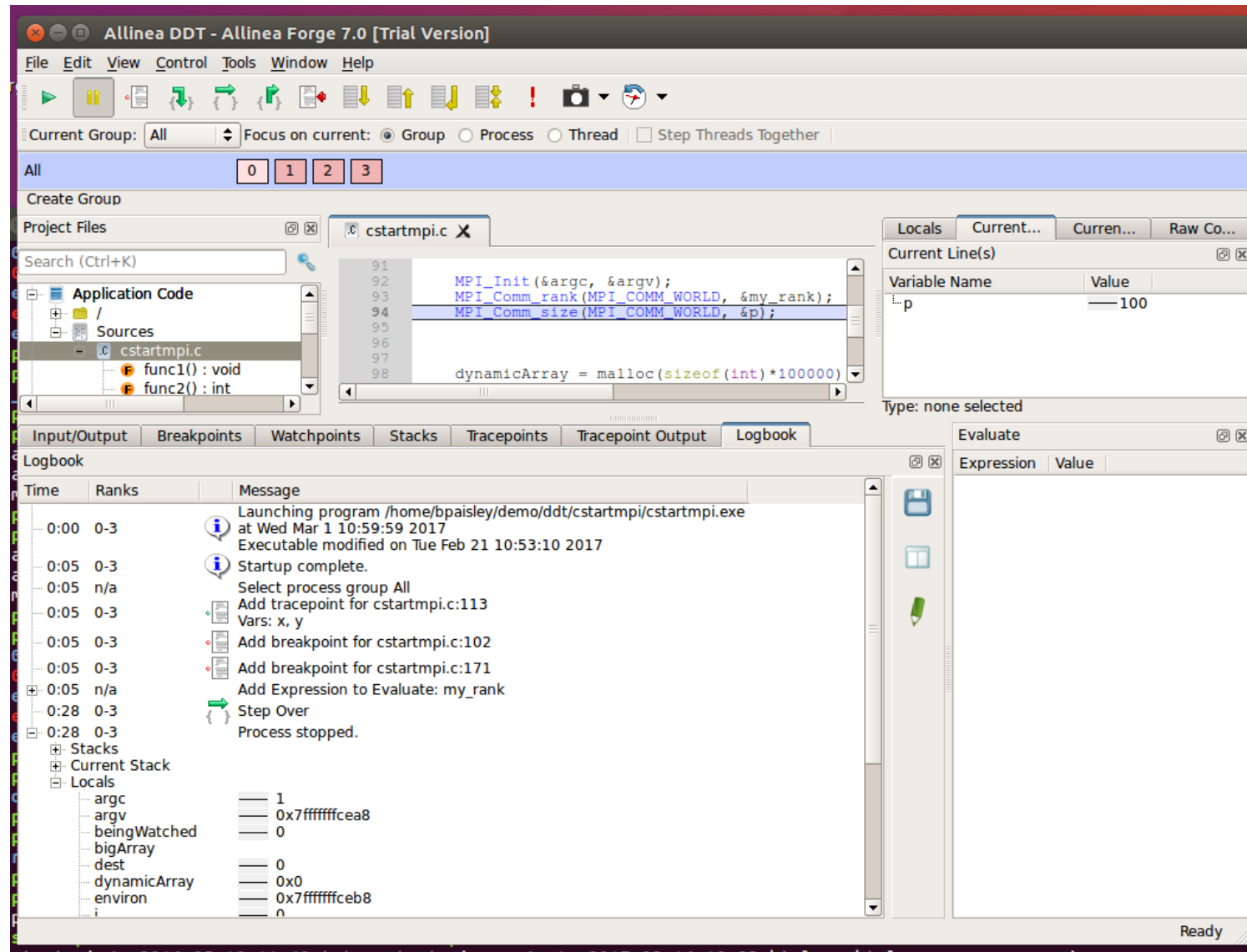
- **A powerful feature...:**
 - Forbids read/write on guard pages throughout the whole execution
(because it overrides C Standard Memory Management library)
- **... to be used carefully:**
 - Kernel limitation: up to 32k guard pages max (“mprotect fails” error)
 - Beware the additional memory usage cost

The screenshot displays the Allinea DDT 4.2.1-36484 debugger interface. The main window shows the source code of `xyzpart.c`, with line 556 highlighted: `my picks[i] = allpicks[i*ntsamples/npes];`. The left pane shows the Project Files tree with `xyzpart.c` selected. The right pane shows the Locals window with variables `allpicks`, `i`, `my picks`, `npes`, and `ntsamples`. The bottom pane shows the Stacks window with the call stack for the current thread. A tooltip is visible over the expression `i * ntsamples`, showing the value `-212322546` and the range from `-2147259746` to `-12282046`.

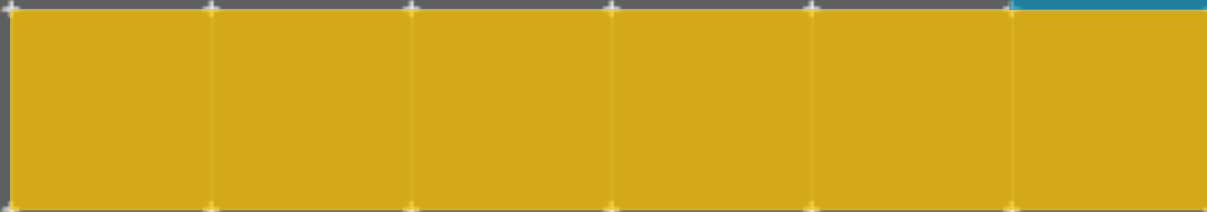
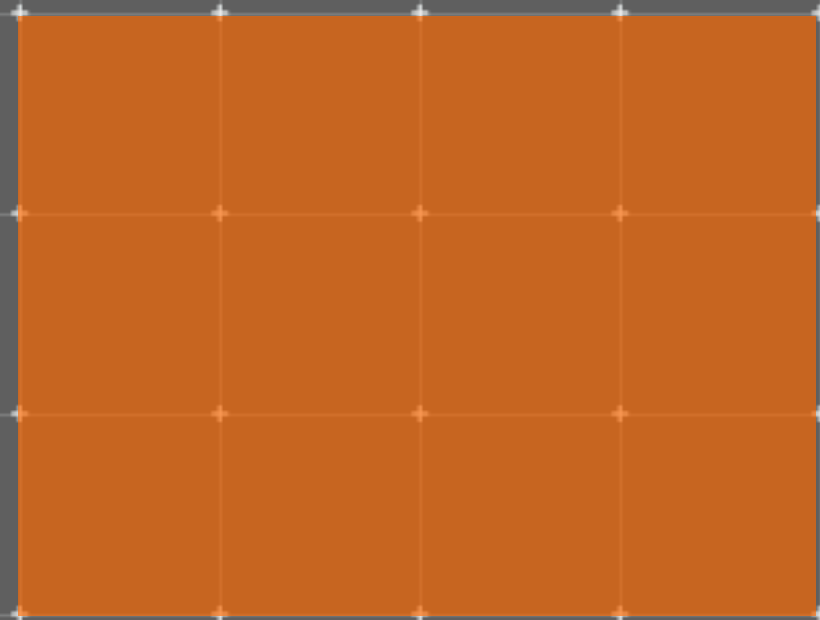
New Bugs from Latest Changes



Track Your Changes in a Logbook



Arm DDT Demo



Python Debugging Now available

Arm DDT - Arm Forge 20.1.1

File Edit View Control Tools Window Help

Current Group: All Focus on current: Group Process Thread Step Threads Together

Project Files

Search (Ctrl+K)

Application Code

Headers

Sources

```
127 print("Info: [Rank{}] Creating matrices".format(rank))
128 my_A = np.random.normal(size=(my_size, my_size)).astype(np.float64)
129 my_B = np.random.normal(size=(my_size, my_size)).astype(np.float64)
130 my_C = np.zeros_like(my_A)
131
132 # Sending matrix
133 print("Info: [Rank{}] Sending matrices".format(rank))
134 for i in range(1, nproc):
135     my_A_slice = my_A[i * slice_size:(i + 1) * slice_size, :].copy(order=
136     comm.Send(my_A_slice, i, MAT_A_TAG)
137     comm.Send(my_B, i, MAT_B_TAG)
138
139 else:
140     # Receiving matrix
141     print("Info: [Rank{}] Receiving matrices".format(rank))
142     my_A = np.empty((slice_size, my_size), np.float64, 'C')
143     my_B = np.empty((my_size, my_size), np.float64, 'C')
144     my_C = np.zeros_like(my_A)
145
146 comm.Recv(my_A, MASTER_RANK, MAT_A_TAG)
147 comm.Recv(my_B, MASTER_RANK, MAT_B_TAG)
```

Locals

Name	Value
my_size	512
solver	"pyloop"
input_fn_pre	None
output_fn_pre	"res"
comm	<mpi4py.MPI.Intracomm at remote...
nproc	8
rank	0
slice_size	64
my_A	<numpy.ndarray at remote 0x2aaac...
my_B	<numpy.ndarray at remote 0x2aaac...

Input/Output Breakpoints Watchpoints Stacks (All) Tracepoints Tracepoint Output Logbook Evaluate

Stacks (All)

Processes	Function
8	<module> (allinea_ddt_trace.py:95)
8	main (allinea_ddt_trace.py:87)
8	<module> (mmprod.py:203)
1	main (mmprod.py:130)
7	main (mmprod.py:143)

mmprod.py:143

7 Processes: ranks 1-7

Evaluate

Name	Value
rank	0
my_A.tolist()	Python list of length 512
[0]	Python list of length 512
[1]	Python list of length 512
[2]	Python list of length 512
[3]	Python list of length 512
[0]	-3.0757148501162277
[1]	-1.3650903487586459
[2]	-0.8629396574226358
[3]	1.8290785164642236
[4]	1.012420520040720

Ready Connected to: (via tunnel) thetalogin6:4201 -> thetamom1

module load
intelpython36

module load
datascience/
mpi4py-3.0.2

ddt --connect aprun -n 8 python %allinea_python_debug% ./mmprod.py -s pyloop -o res -n 512

Five great things to try with Alinea DDT

Input/Output	Breakpoints	Watchpoints	Tracepoints	Tracepoint Output	Stacks (All)
Tracepoint Output					
Tracepoint	Processes	Values logged			
vhone 90.85	976, ranks 12,14-17,22-23,12...	mype 2170-3527 jcol 2-43 mod	pey		
vhone 90.81	960, ranks 12,14-17,22-23,12...	ks 1 kmax	pez		
vhone 90.85	942, ranks 12,14-17,22-23,12...	mype 2170-3527 jcol 2-43 mod	pey		
vhone 90.81	920, ranks 12,14-17,22-23,12...	ks 1 kmax	pez		
vhone 90.85	918, ranks 12,14-17,22-23,12...	mype 2170-3527 jcol 2-43 mod	pey		
vhone 90.81	898, ranks 12,14-17,22-23,12...	ks 1 kmax	pez		
vhone 90.85	884, ranks 12,14-17,22-23,12...				
vhone 90.81	880, ranks 12,14-17,22-23,12...				

The scalable print alternative

```

for (i = 0 ; i < SIZE M; i++)
  for (j = 0 ; j < SIZE N; j++)
    C[i][j] = 0;

for (i = 0 ; i < SIZE M; i++)
  for (j = 0 ; j < SIZE N; j++)
    for (k = 0 ; k < SIZE 0; k++)
      C[i][j] += A[i][k] * B[k][j];
    
```

Stop on variable change

```

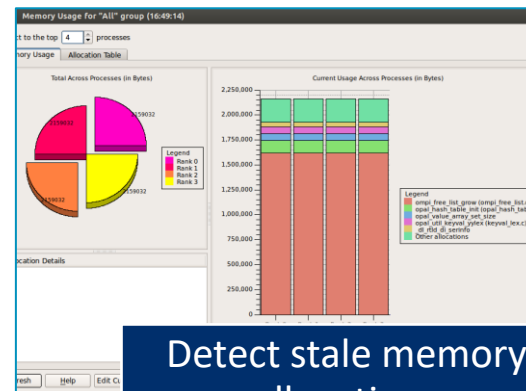
43
44 else
45 {
46   test=-1;
47 }
48 void func3()
49 {
50   void* i = (void*) 1;
51   while(i++ || !i)
52     free((void*)i);
    
```

Static analysis warnings on code errors

```

&& !strcmp(argv[i], "crash")) {
0;
s", *(char**)argv[i]);
ll se
    
```

Detect read/write beyond array bounds



Detect stale memory allocations

Arm DDT cheat sheet

Load the environment module

- \$ module load **forge/20.1.1**
- \$ module unload **xalt**
- \$ module unload **darshan**

Prepare the code

- \$ cc **-O0 -g** myapp.c -o myapp.exe

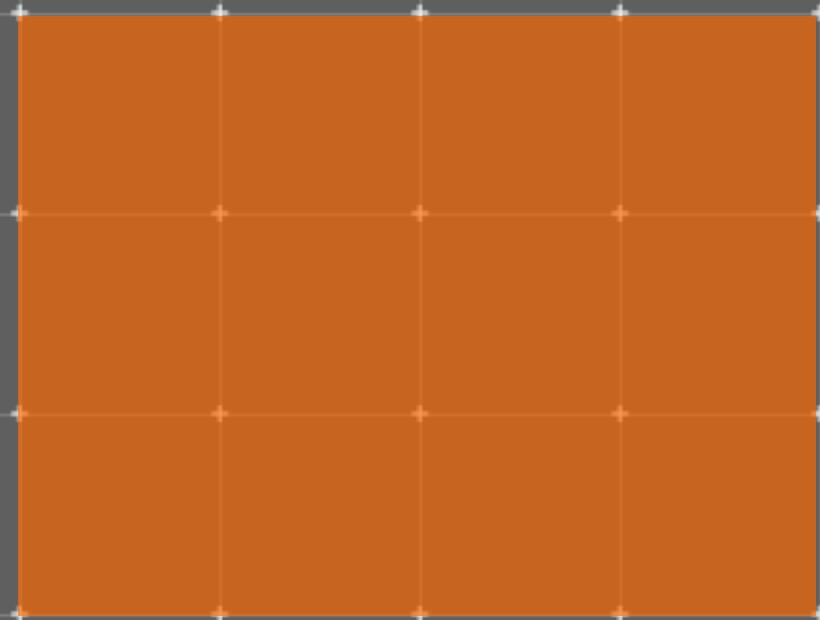
Start Arm DDT in interactive mode

- \$ **ddt** aprun -n 8 ./myapp.exe arg1 arg2

Or use the reverse connect mechanism

- On the login node:
 - \$ ddt &
- (or use the remote client) <- **Preferred method**
- Then, edit the job script to run the following command and submit:
 - **ddt --connect** aprun -n 8 ./myapp.exe arg1 arg2

Profiling with MAP



Arm MAP – The Profiler



Small data files



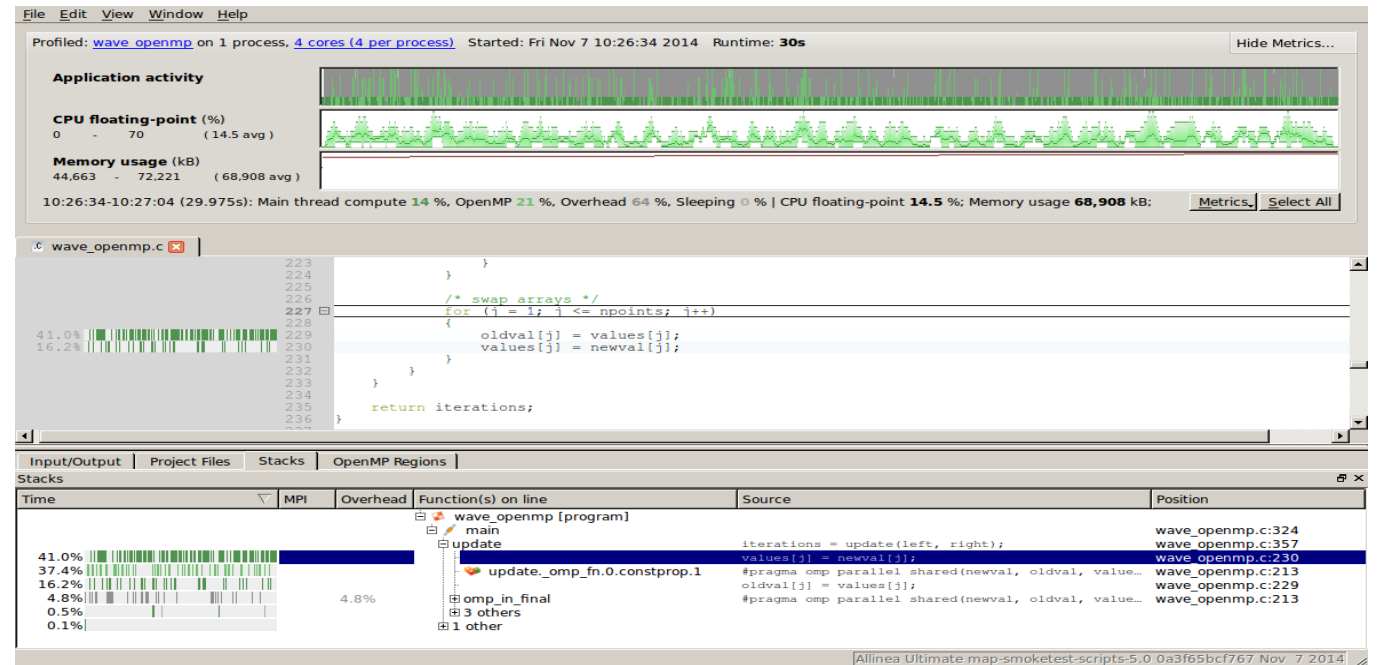
<5% slowdown



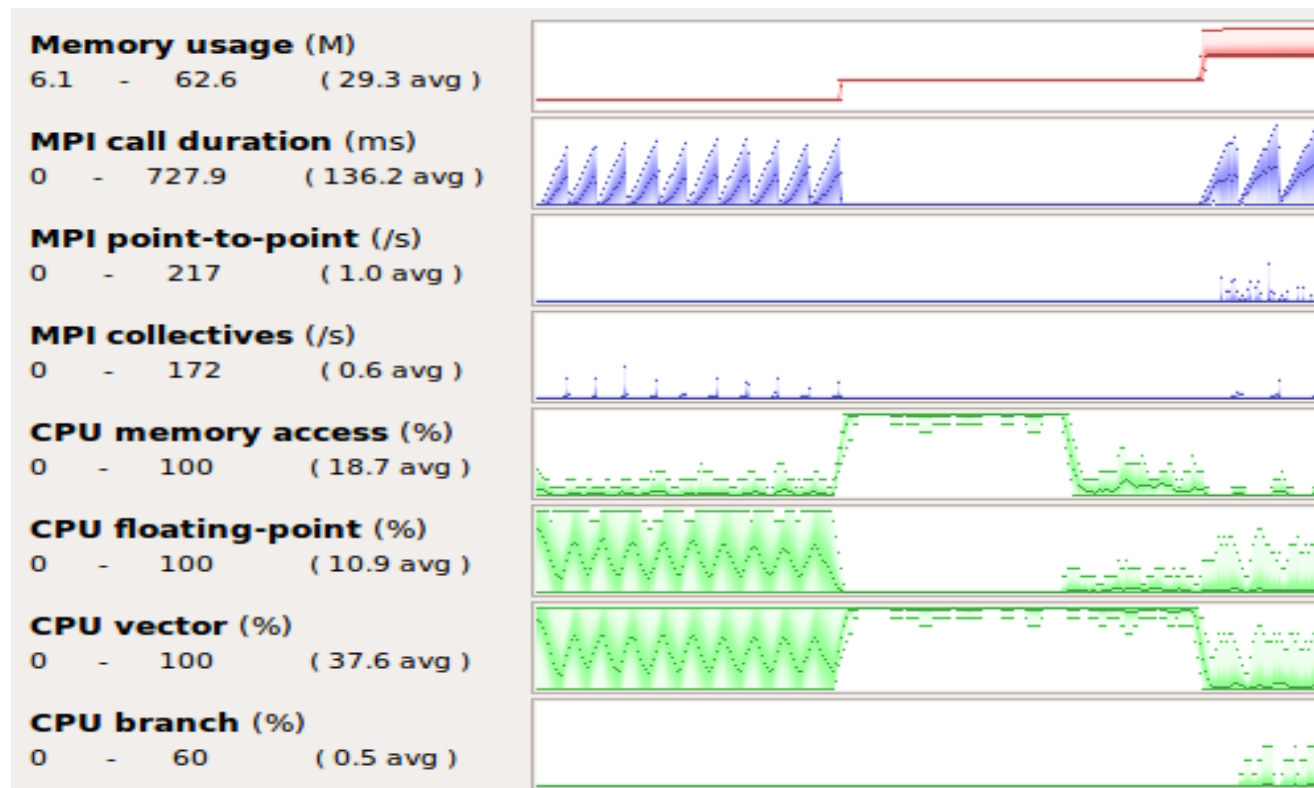
No instrumentation



No recompilation



Glean Deep Insight from our Source-Level Profiler



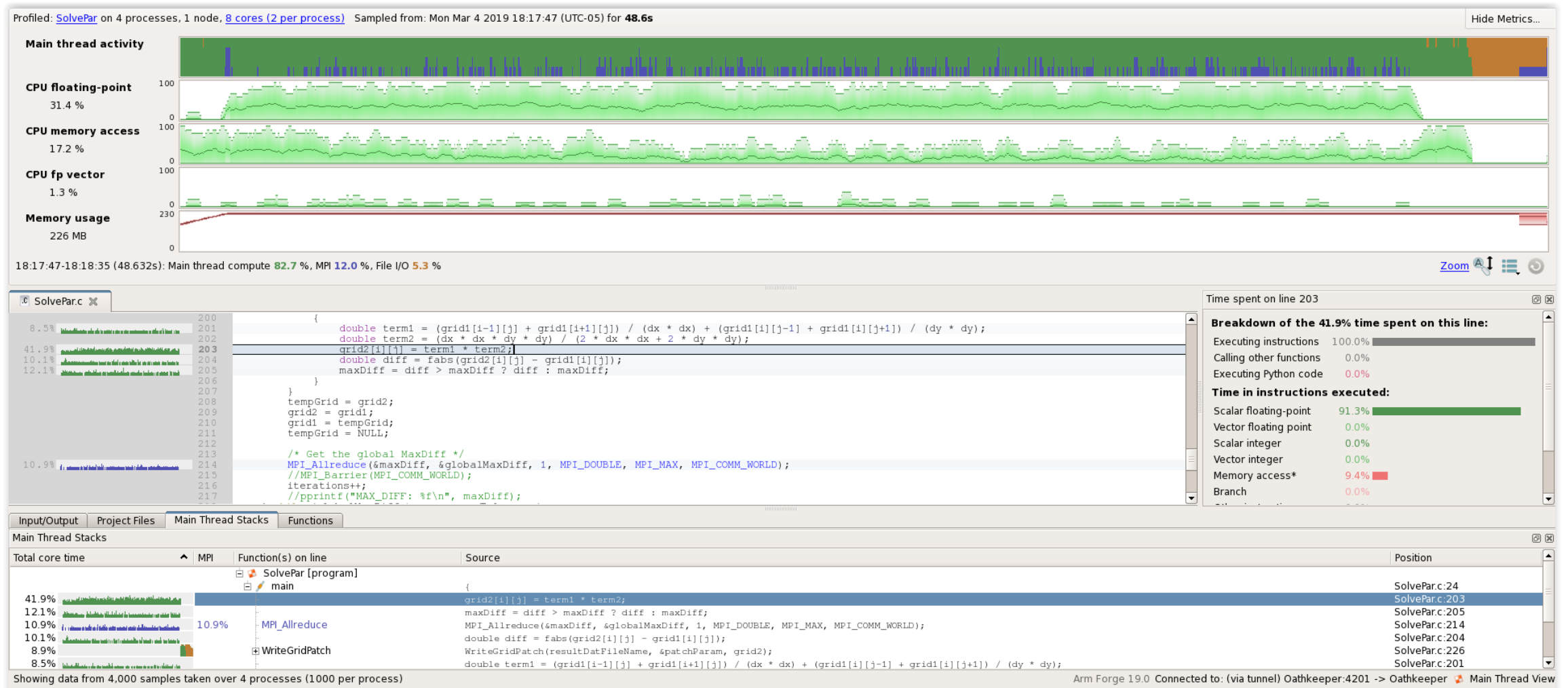
Track memory usage across the entire application over time

Spot MPI and OpenMP imbalance and overhead

Optimize CPU memory and vectorization in loops

Detect and diagnose I/O bottlenecks at real scale

Profile of 2d Laplace Solver with Jacobi Iteration



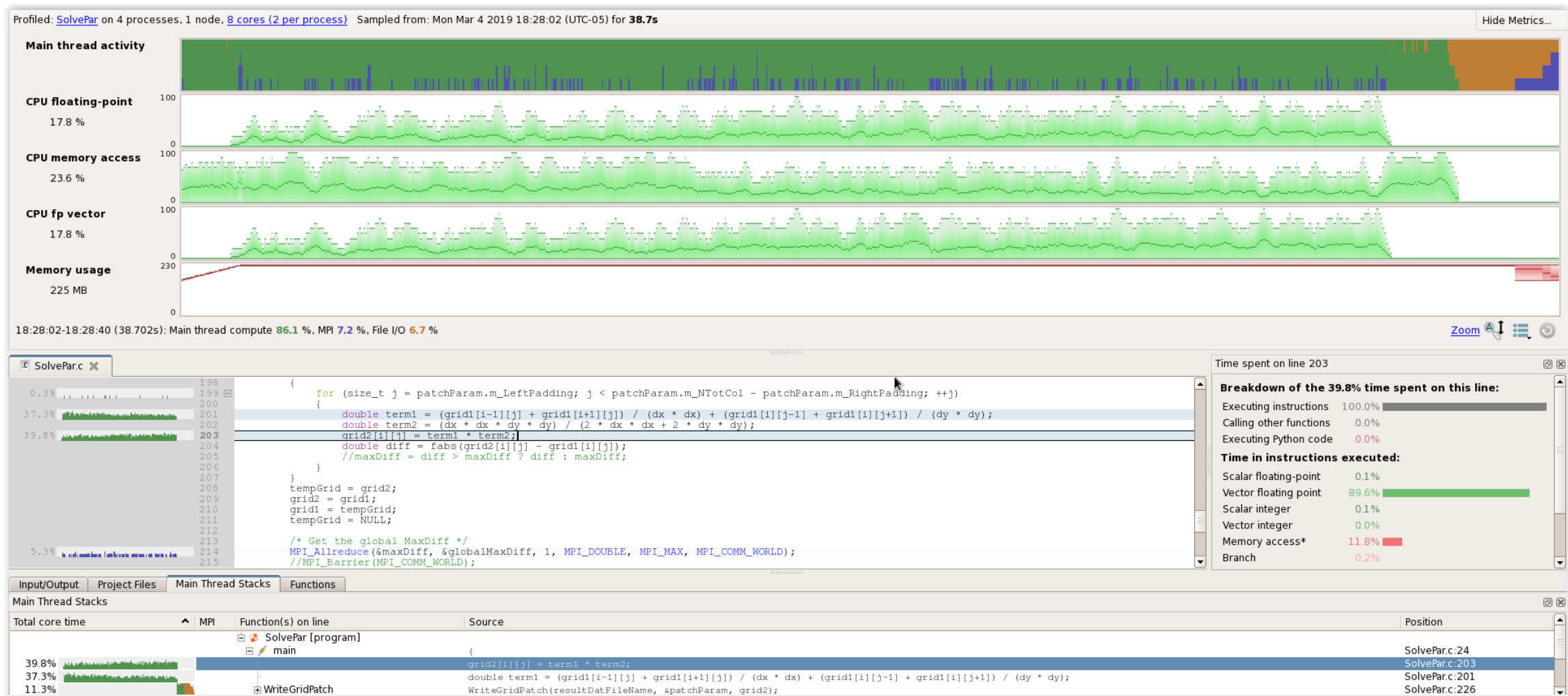
Tracking Largest Change

```
// Compare newly computed value with old value
diff = fabs(grid2[i][j] - grid1[i][j]);

// Track largest change between new and old values
maxDiff = diff > maxDiff ? Diff : maxDiff;

If (diff > maxDiff)
    then maxDiff= diff;
Else
    maxDiff = maxDiff;
```

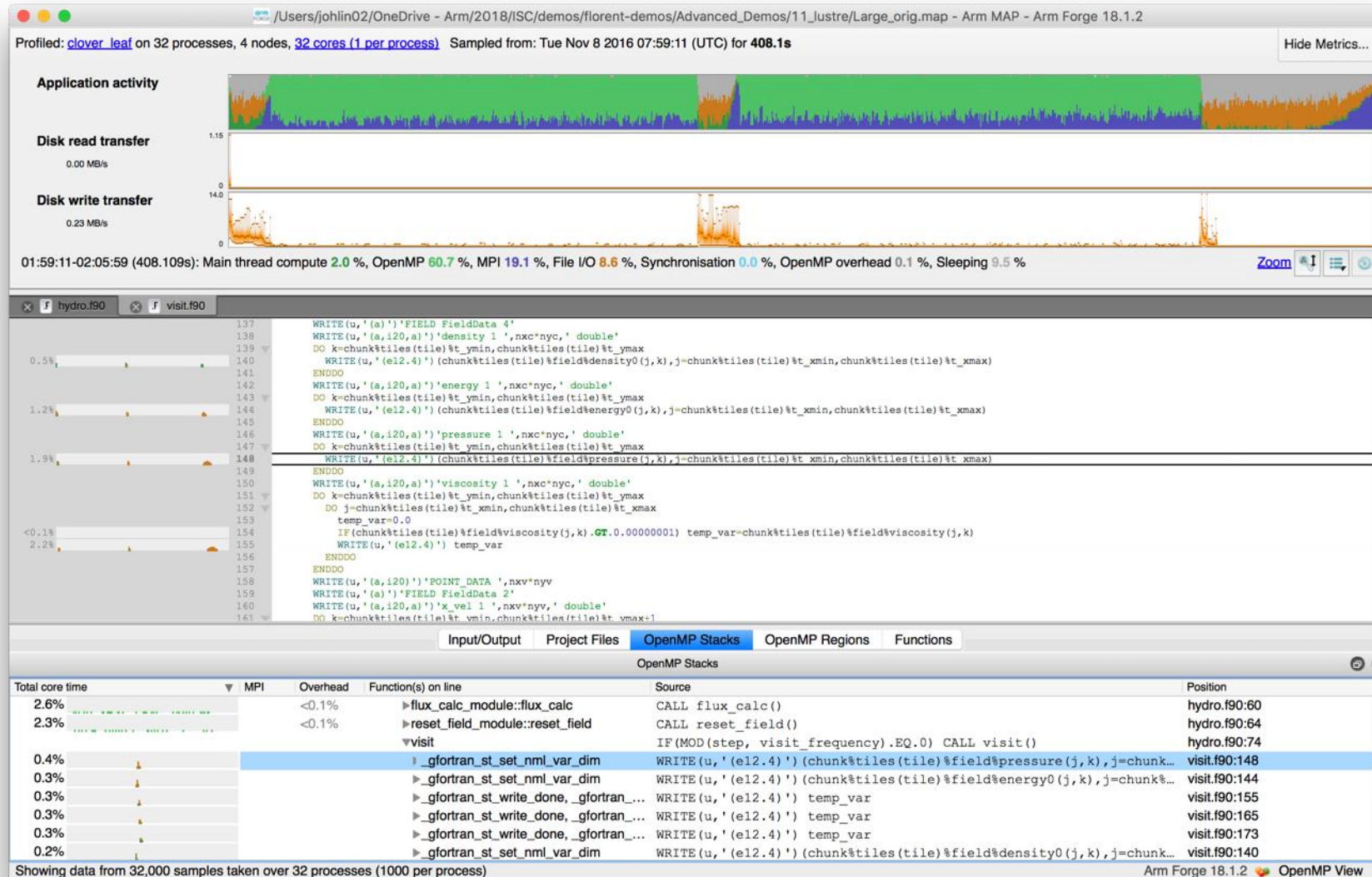
Conditional Removal from Innermost Loop



20 % faster, also operation is now vectorized

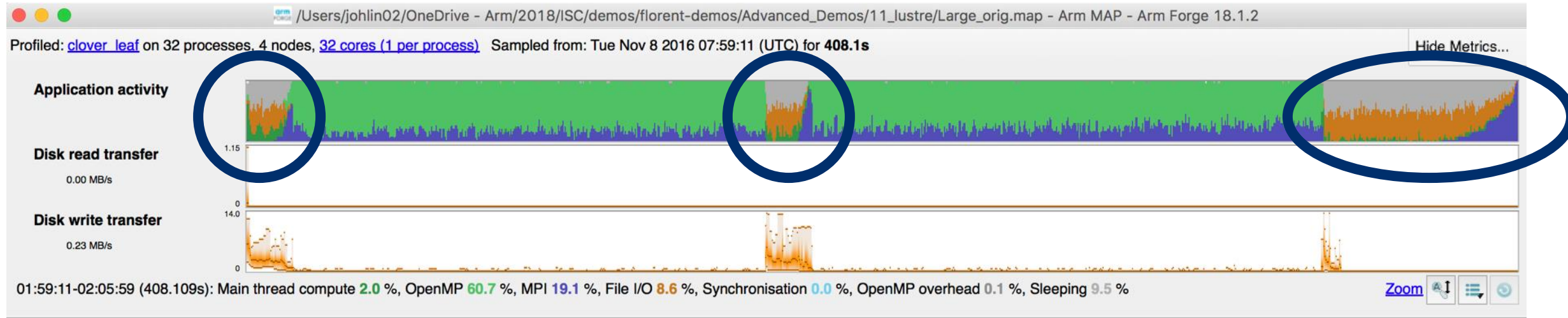
Initial profile of CloverLeaf shows surprisingly unequal I/O

Each I/O operation should take about the same time, but it's not the case.



Symptoms and causes of the I/O issues

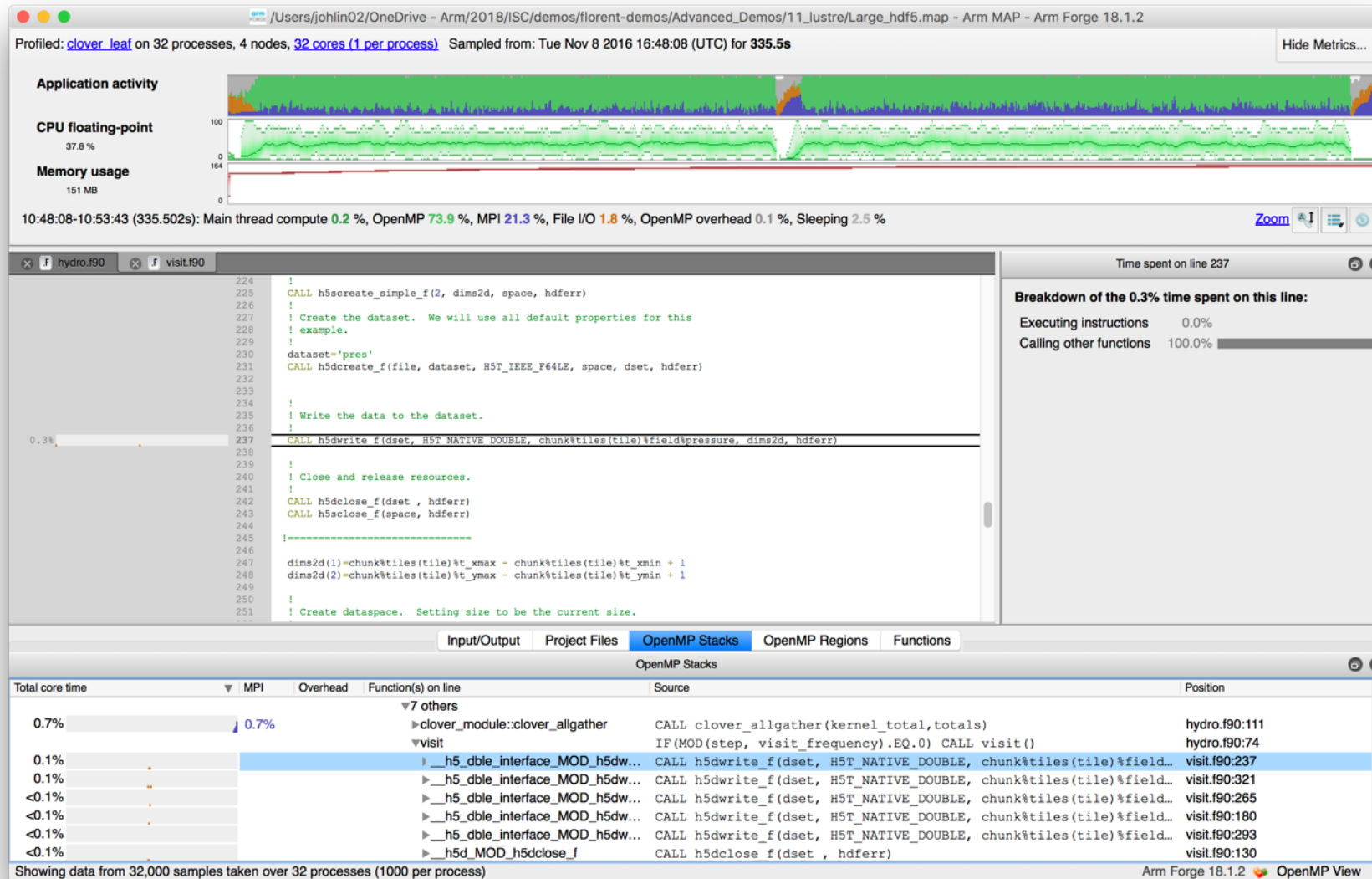
Sub-optimal file format and surprise buffering.



- Write rate is less than 14MB/s.
- Writing an ASCII output file.
- Writes not being flushed until buffer is full.
 - Some ranks have much less buffered data than others.
 - Ranks with small buffers wait in barrier for other ranks to finish flushing their buffers.

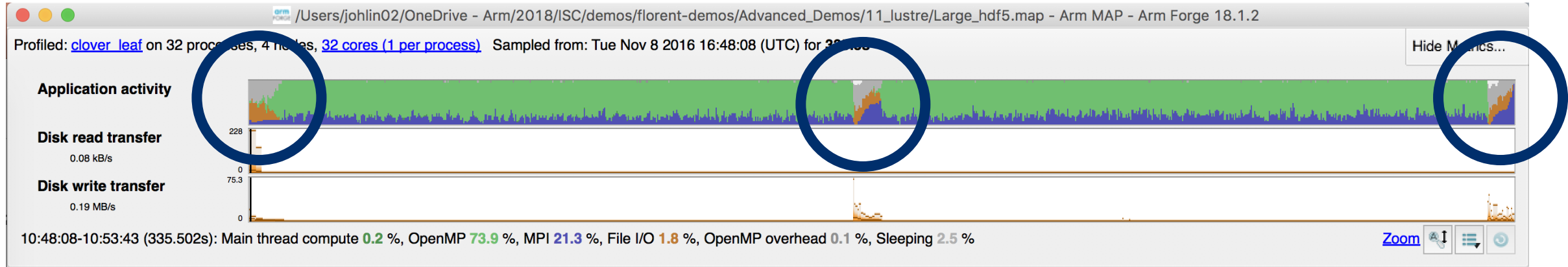
Solution: use HDF5 to write binary files

Using a library optimized for HPC I/O improves performance and portability.



Solution: use HDF5 to write binary files

Using a library optimized for HPC I/O improves performance and portability.



- Replace Fortran write statements with HDF5 library calls.
 - Binary format reduces write volume and can improve data precision.
 - Maximum transfer rate now 75.3 MB/s, over 5x faster.
- Note MPI costs (blue) in the I/O region, so room for improvement.

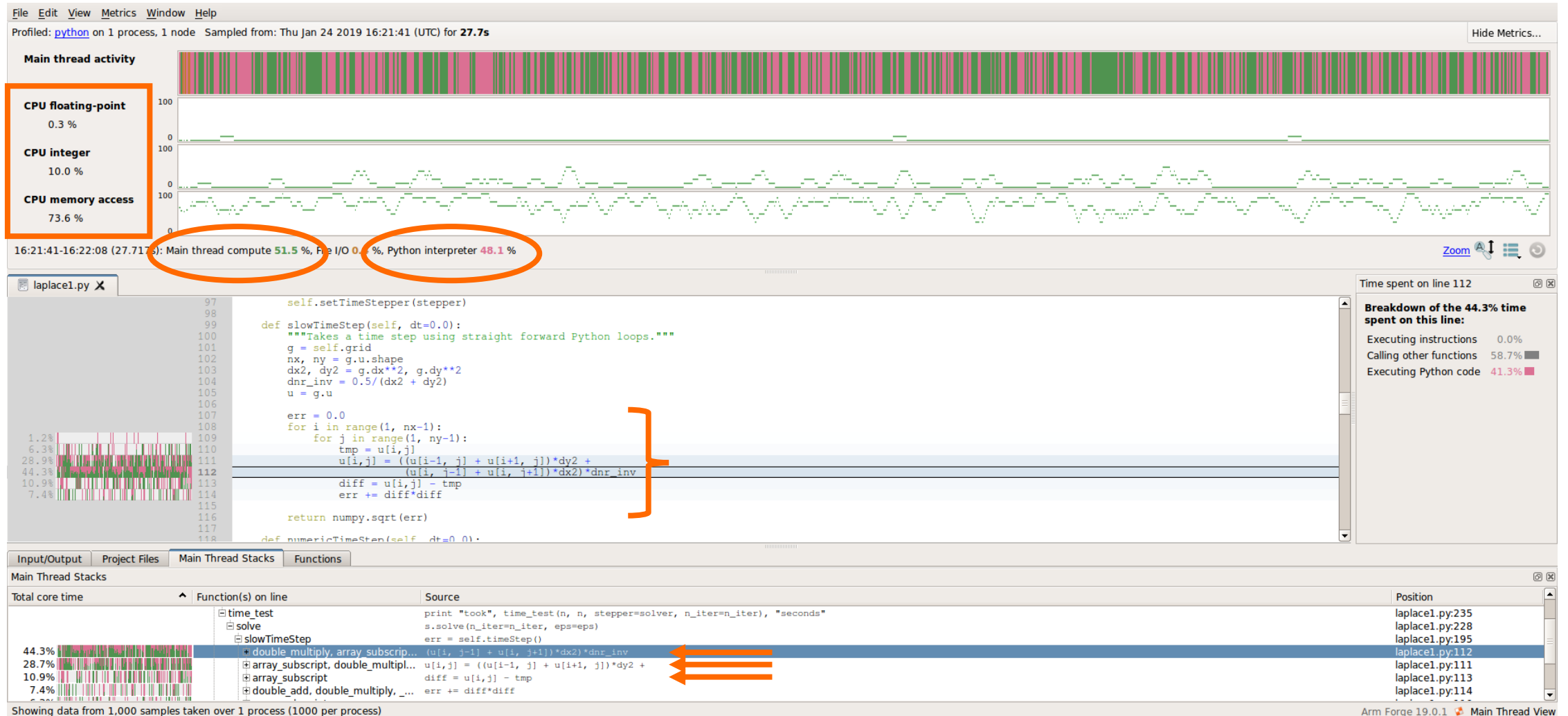
Arm MAP: Python profiling

- Launch command
 - \$ **python** ./laplace1.py slow 100 100
- Profiling command
 - \$ **map --profile python** ./laplace1.py slow 100 100
 - --profile: non-interactive mode
 - --output: name of output file
- Display profiling results
 - \$ **map** laplace1.map

Laplace1.py

```
[...]
err = 0.0
for i in range(1, nx-1):
    for j in range(1, ny-1):
        tmp = u[i,j]
        u[i,j] = ((u[i-1, j] + u[i+1, j])*dy2 +
                  (u[i, j-1] + u[i, j+1])*dx2)*dnr_inv
        diff = u[i,j] - tmp
        err += diff*diff
return numpy.sqrt(err)
[...]
```

Naïve Python loop (laplace1.py slow 100 1000)




Optimizing computation on NumPy arrays

Naïve Python loop

```
err = 0.0
for i in range(1, nx-1):
    for j in range(1, ny-1):
        tmp = u[i,j]
        u[i,j] = ((u[i-1, j] + u[i+1, j])*dy2 +
                  (u[i, j-1] + u[i, j+1])*dx2)*dnr_inv
        diff = u[i,j] - tmp
        err += diff*diff
return numpy.sqrt(err)
```

NumPy loop

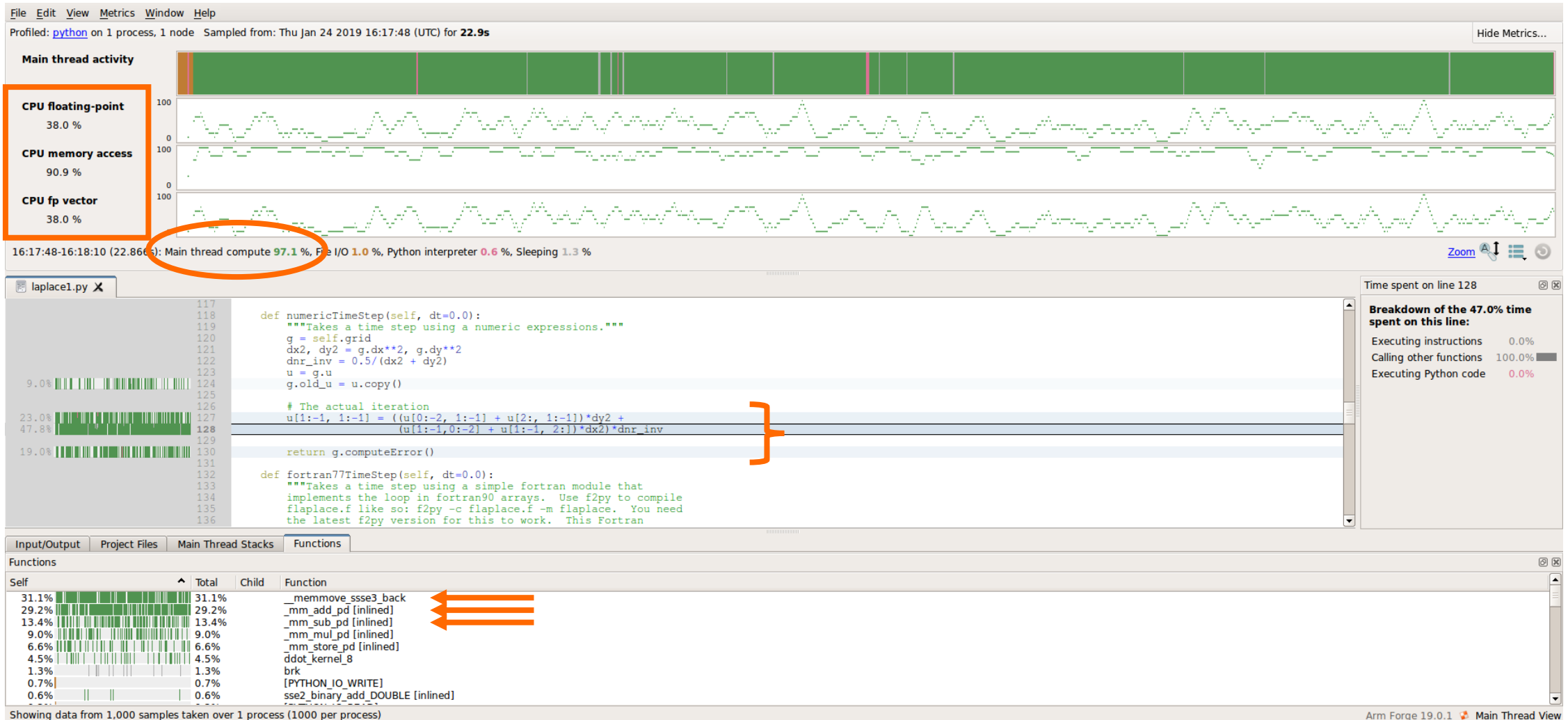


```
u[1:-1, 1:-1] =
    ((u[0:-2, 1:-1] + u[2:, 1:-1])*dy2 +
     (u[1:-1, 0:-2] + u[1:-1, 2:])*dx2)*dnr_inv

return g.computeError()
```


NumPy array notation (laplace1.py numeric 1000 1000)

This is 10 times more iterations than was computed in the previous profile



Arm MAP cheat sheet

Load the environment module (manually specify version)

- `$ module load forge/20.1.1`

Set the default debugger to gdb 8.2

- `$ export ALLINEA_FORCE_DEBUGGER=gdb-82`

Unload Darshan module (It wraps MPI calls which cannot be used with MAP)

- `$ module unload darshan`

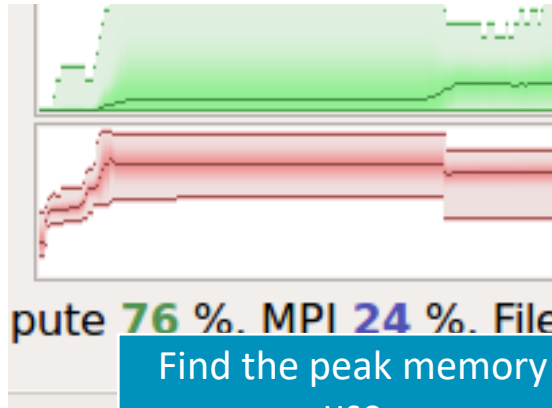
Compile the code with dynamic linking

- `$ cc -O3 -g -dynamic myapp.c -o myapp.exe`
- Edit the job script to run Arm MAP in “profile” mode
- `$ map --profile aprun -n 8 ./myapp.exe arg1 arg2`

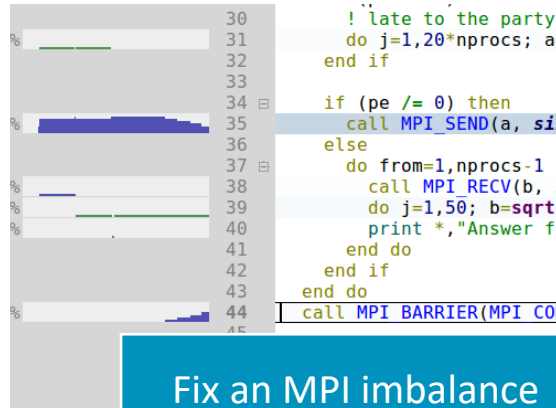
Open the results

- On the login node:
 - `$ map myapp_Xp_Yn_YYYY-MM-DD_HH-MM.map`
- (or load the corresponding file using the remote client connected to the remote system or locally)

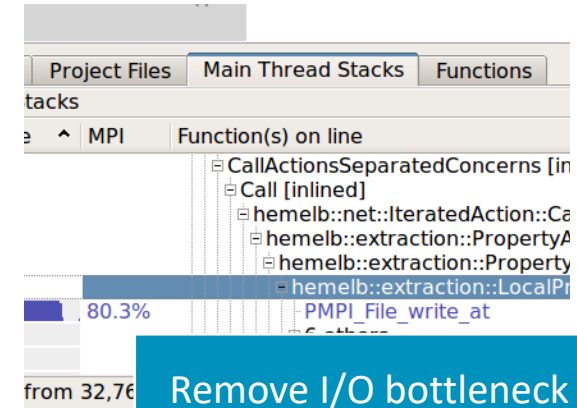
Six Great Things to Try with Alinea MAP



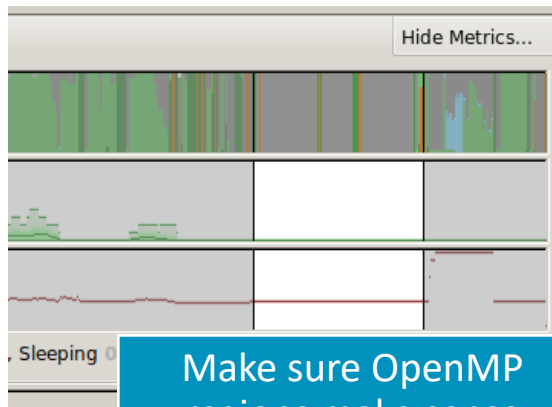
Find the peak memory use



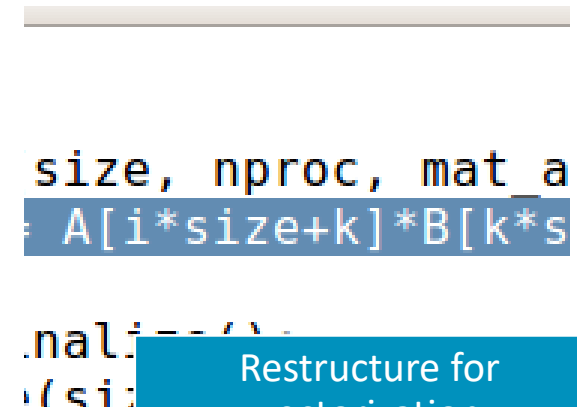
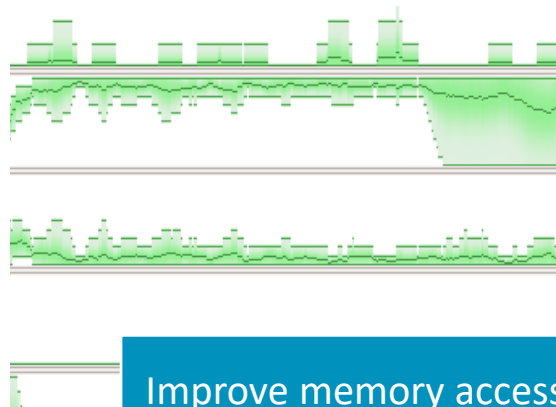
Fix an MPI imbalance



Remove I/O bottleneck

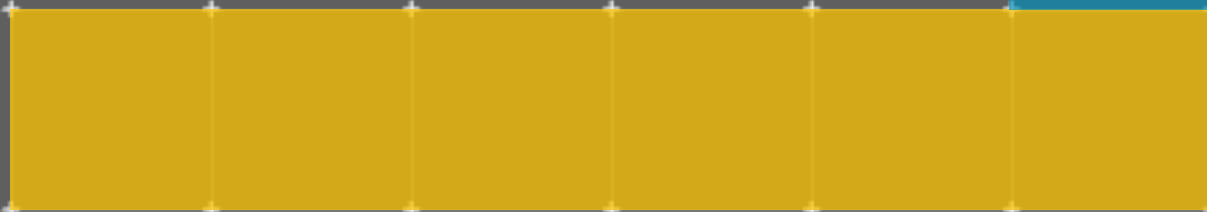
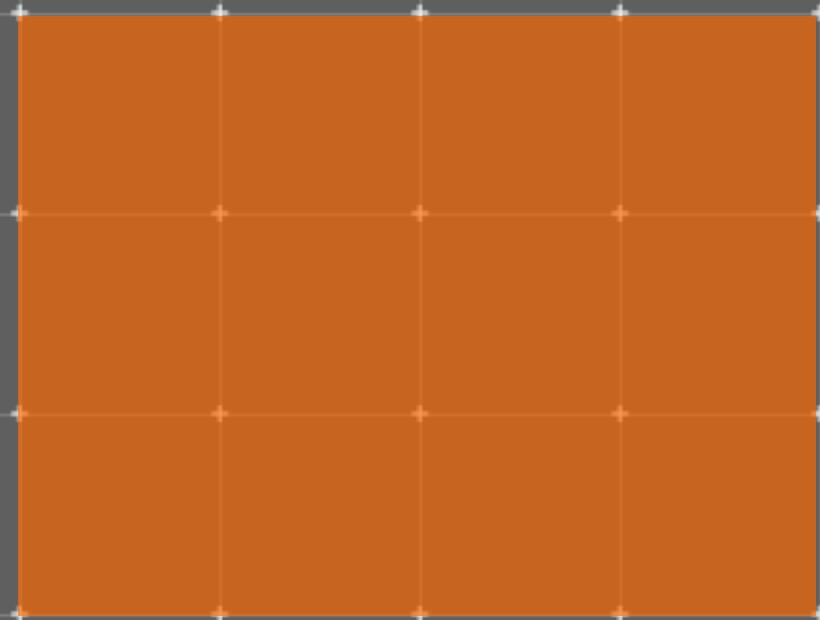


Make sure OpenMP regions make sense



Restructure for vectorization

Theta Specific Settings



Configure the remote client

Install the Arm Remote Client

- Go to : <https://developer.arm.com/products/software-development-tools/hpc/downloads/download-arm-forge>

Connect to the cluster with the remote client

- Open your Remote Client
- Create a new connection: Remote Launch → Configure → Add
 - Hostname: <username>@theta.alcf.anl.gov
 - Remote installation directory:

`/soft/debuggers/soft/debuggers/forge-20.1.1-2020-08-02`

Using the Windows remote client

Due to the higher security measures on ALCF systems it is necessary to upgrade the plink.exe that is shipped with the Forge Client

Visit <https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html> and download the latest plink.exe

Once saved, copy and paste this file into the **libexec** folder of the Arm Forge Client installation directory

Workaround for Forge 20.1

The scheduler causes ddt/map to think that resources are not available and results in the following error: **apsched: request exceeds max nodes, alloc**

The workaround is to set the environment variable `ALLINEA_USE_PSEUDO_TTY` to `no`

For Linux/Mac users, you can do this in the terminal on your local machine i.e.

```
export ALLINEA_USE_PSEUDO_TTY=no
```

For Windows users see the following slide

Workaround for Forge 20.1

The image shows a Windows 10 desktop with the Start menu open. The 'env' search results are displayed, with 'Edit the system environment variables' highlighted. This is labeled 'Step 1'. The 'System Properties' window is open, showing the 'Advanced' tab. The 'Environment Variables...' button is highlighted, labeled 'Step 2'. The 'Environment Variables' window is open, showing the 'User variables for ryahul01' tab. The 'Path' variable is selected, and the 'New...' button is highlighted, labeled 'Step 3'. The 'New System Variable' window is open, showing the 'Variable name' field with 'ALLINEA_USE_PSEUDO_TTY' and the 'Variable value' field with 'no'. The 'New' button is highlighted, labeled 'Step 4'. The 'arm' logo is in the bottom right corner.

Step 1

Step 2

Step 3

Step 4

arm

Static Linking Extra Steps

To enable advanced memory debugging features in DDT with static binaries, you must link explicitly against our memory debugging libraries

Simply add the link flags to your Makefile, or however appropriate

```
lflags = -L/soft/debuggers/ddt/lib/64 -Wl,--undefined=malloc -ldmallocth -Wl,--allow-multiple-definition
```


Profiling on Theta

Although static binaries are created by default on Theta, it is recommended to build dynamic executables for profiling purposes with the compiler flag **-dynamic**

If you get library missing errors, reload the intel module

```
module unload intel
```

```
module load intel
```

If you get GdbmiParser errors set the following environment variable

```
export ALLINEA_FORCE_DEBUGGER=gdb-82
```

Examples for hands-on session

Examples are available at `/projects/ATPESC_Instructors/arm_forge/`

```
mkdir -p /projects/ATPESC2020/$USER
```

```
cp -r /projects/ATPESC_Instructors/arm_forge/arm-hpc-examples /projects/ATPESC2020/$USER
```

```
cd /projects/ATPESC2020/$USER/arm-hpc-examples
```

README files are available for each example

Questions?

Thank You!

Danke!

Merci!

谢谢!

ありがとう!

Gracias!

Kiitos!

감사합니다

धन्यवाद

arm